**Computer Networks and Programming**

**(ECE 5650)**

**Project 3**

**Web page Downloader using multi-threading**

**Rajeev Bhupathiraju (gd4707@wayne.edu)**

**Sai Sundeep Innamuri (fx3898@wayne.edu)**

Wayne State University

Winter 2017

**Introduction**

**HTML (Hyper Text Markup Language):**

Hyper Text Markup Language (HTML) is the standard [markup language](https://en.wikipedia.org/wiki/Markup_language) for creating [web pages](https://en.wikipedia.org/wiki/Web_page) and [web applications](https://en.wikipedia.org/wiki/Web_application). With [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS), and [JavaScript](https://en.wikipedia.org/wiki/JavaScript), it forms a triad of cornerstone technologies for the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web). [Web browsers](https://en.wikipedia.org/wiki/Web_browser) receive HTML documents from a [webserver](https://en.wikipedia.org/wiki/Webserver) or from local storage and render them into multimedia web pages. HTML describes the structure of a web page [semantically](https://en.wikipedia.org/wiki/Semantic) and originally included cues for the appearance of the document.

**Objective**

The aim of the project is:

1. Enhancing webpage downloader in project 2 by supporting multithreading and obtaining all required features

2. Comparing the performance of multi-threaded web page downloader with previous project and tabulating the results and plot graphs

**Code:**



##########################################################################################

# Course : Computer Networks and Programming, Winter 2017

# project : project 3

# author : Sai Sundeep Innamuri - 004507888, Rajeev Bhupathiraju - 004570122

# file name : project3.py

# date Written : Mar 26 2017

##########################################################################################

from HTMLParser import HTMLParser

from socket import \*

import sys

import os

import time

from datetime import datetime

from datetime import timedelta

import threading

HTTPGETRequestFormat = "GET {} HTTP/1.1\nHost: {}\n\n"

httpPort = 80

# serverSocket = socket(AF\_INET, SOCK\_STREAM)

domain\_names = ('com','org','edu','in','uk')

# create a subclass and override the handler methods

class MyHTMLParser(HTMLParser):

title = None

srcs= list()

next\_tag\_data\_type = None

def getAttrValue(self, attrs, key):

for attr in attrs:

if attr[0] == key: return attr[1]

return None

def handle\_starttag(self, tag, attrs):

if tag == 'title': self.next\_tag\_data\_type = 'title'

elif tag == 'img':

src = self.getAttrValue(attrs, "src")

if src not in self.srcs:

self.srcs.append(self.getAttrValue(attrs,"src"))

else:

src = self.getAttrValue(attrs,"src")

href= self.getAttrValue(attrs,"href")

attrval = None

if src is not None: attrval = src

if href is not None: attrval = href

if attrval is None: return

if attrval.find('http') != -1: return

# print attrval

self.srcs.append(attrval)

def handle\_data(self, data):

if self.next\_tag\_data\_type == 'title' :

self.title = data.replace(" ","\_").replace("\n","").replace(":", "\_")

self.next\_tag\_data\_type = None

def writeFile(path, file, body):

if not os.path.exists(path):

# print os.path.basename(path)

os.mkdir(path)

elif file.rfind("/") != -1:

os.mkdir(path + "/" + file[:file.rfind("/")])

htmlfile = open(path + '/' + file, "wb")

htmlfile.write(body)

htmlfile.close()

#function to seperate header and body from a html response

def getHeadersNBody(response):

headers = response[: response.index("\r\n\r\n")]

body = response[response.index("\r\n\r\n") + 4:]

headers = headers.split("\r\n")

headersDict = dict()

headersDict["status"]=headers[0][9:12]

headers = headers[1:]

# print headers

#making a dictonary object for headers

for header in headers:

try:

sp = header.split(": ")

headersDict[sp[0]] = sp[1]

except Exception, e:

# print "cannot parse header" + header

None

return headersDict, body

#function to send and receive HTTP request

def sendRecvHTTPReq(host, HTTPRequest):

# print HTTPRequest[:-2]

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.connect((host, httpPort))

serverSocket.settimeout(1)

#sending the http request

serverSocket.send(HTTPRequest)

#receving the response

response = serverSocket.recv(10000)

headers, body = getHeadersNBody(response)

# print "status : " + headers["status"]

# handling differt status codes

if headers["status"] in ('302', '301', '307' ):

# print "Encountered redirection. Not sipported!"

serverSocket.close()

raise Exception

# protocal, host, request = extractDetailFromURL(headers["Location"])

# serverSocket.close()

# headers, body = sendRecvHTTPReq(host, HTTPGETRequestFormat.format(request, host))

# return headers, body

elif headers["status"] != '200':

# print response

# print "Some error in request. HTTP error code " + headers["status"]

# print "Exiting"

serverSocket.close()

raise Exception

# sys.exit(2)

#receving more messages if the entire conteent is not received in one request

contentReceived = len(body)

if "Content-Length" in headers.keys():

# print "content is chunked. downloading in Iterations, Content length in header keys"

while (contentReceived < int(headers["Content-Length"])):

body = body + serverSocket.recv(10000)

contentReceived = len(body)

elif "Transfer-Encoding" in headers.keys() and headers["Transfer-Encoding"] == "chunked":

# print "content is chunked. downloading in Iterations. no content length"

while (1):

newChunk = serverSocket.recv(10000)

if newChunk == "": break

body = body + newChunk

time.sleep(1)

serverSocket.close()

return headers,body

def checkForDomainInString(request):

for domain in domain\_names:

if domain\_names.find('.'+domain) != -1 : return domain

else: continue

return None

thread\_list = []

srcs\_to\_replace = dict()

def objectDownloadThread(homepage, initreq, host, folderToSave):

try:

request = initreq

request = request.replace(" ", "%20")

if request[0] != "/":

req = HTTPGETRequestFormat.format(homepage + request, host)

elif request.find("//") != -1:

nprotocal, nhost, nrequest = extractDetailFromURL(request)

req = HTTPGETRequestFormat.format(nrequest, nhost)

else:

req = HTTPGETRequestFormat.format(request, host)

headers, body = sendRecvHTTPReq(host, req)

if initreq[-1] == "/": initreq = initreq[:-1]

if initreq.rfind("/") != -1:

savefile = initreq.replace("/", "\_")

# print "Save file "+ savefile

if initreq not in srcs\_to\_replace.keys(): srcs\_to\_replace[initreq] = savefile

writeFile(folderToSave, savefile, body)

else:

writeFile(folderToSave, initreq, body)

except:

None

# print "Exception, Proceeding wtith next"

#function to download an array of bojects from src tag

def processReqArray(homepage, reqArray, host, folderToSave):

# print reqArray

numObjects = 0

for initreq in reqArray:

t = threading.Thread(target=objectDownloadThread, args=(homepage, initreq, host, folderToSave))

thread\_list.append(t)

for thread in thread\_list:

thread.start()

for i in range (0, len(thread\_list)):

thread\_list[i].join()

sys.stdout.write("\rThread Joined for Object: %d" %(i+1))

sys.stdout.flush()

print ""

return numObjects, srcs\_to\_replace

#to extract details from URL

def extractDetailFromURL(htmlReq):

protocal = htmlReq.split("//")

host = protocal[1].split("/")[0]

request = protocal[1][protocal[1].index('/'):]

protocal = protocal[0]

return protocal, host, request

def main():

parser = MyHTMLParser()

try:

start\_time = datetime.now()

print "start\_time : " + str(start\_time)

#storing the web site request in to a variable

protocal, host, indexReq = extractDetailFromURL(sys.argv[1])

headers,body = sendRecvHTTPReq(host, HTTPGETRequestFormat.format(indexReq, host))

websiteHome = indexReq if indexReq == "/" else indexReq[:indexReq[1:].index("/")+2]

#to parse the HTML files

parser.feed(body)

if os.path.exists(parser.title):

for root, dirs, files in os.walk(parser.title, topdown=False):

for name in files:

os.remove(os.path.join(root, name))

for name in dirs:

os.rmdir(os.path.join(root, name))

else: os.mkdir(parser.title)

# htmlfile = open(parser.title + "/" + parser.title +".html" , "wb")

#to download the images

num\_objects, srcs\_to\_replace = processReqArray(websiteHome, parser.srcs, host, parser.title)

end\_time = datetime.now()

print "end\_time : " + str(end\_time)

for k,v in srcs\_to\_replace.items():

# print "replacing keyy " +k+ " with "+v

body = body.replace(k, v)

# print "writing file : " + parser.title +".html"

writeFile(parser.title, parser.title +".html", body )

print "totaltime : "+ str(timedelta.total\_seconds(end\_time - start\_time))

except Exception,e :

None

print str(e) + " ignored"

finally:

None

if os.path.exists(parser.title):

print ""

print "please check {} folder for downloaded html website".format(parser.title)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Testing procedure:**

1. We have code for project 2 and project 3 in this submission
2. Project 2 code is named project2.py
   1. To run project 2: python project2.py <any URL>
3. Project 3 code with multi-threading is named project3.py
   1. To run project 3: python project3.py <any URL>
4. We have tested with four URL. They are
5. <http://www.ece.eng.wayne.edu/~nabil/>
6. <http://ece.eng.wayne.edu/~hying/>
7. <http://ece.eng.wayne.edu/~apandya/>
8. <http://imgur.com/>

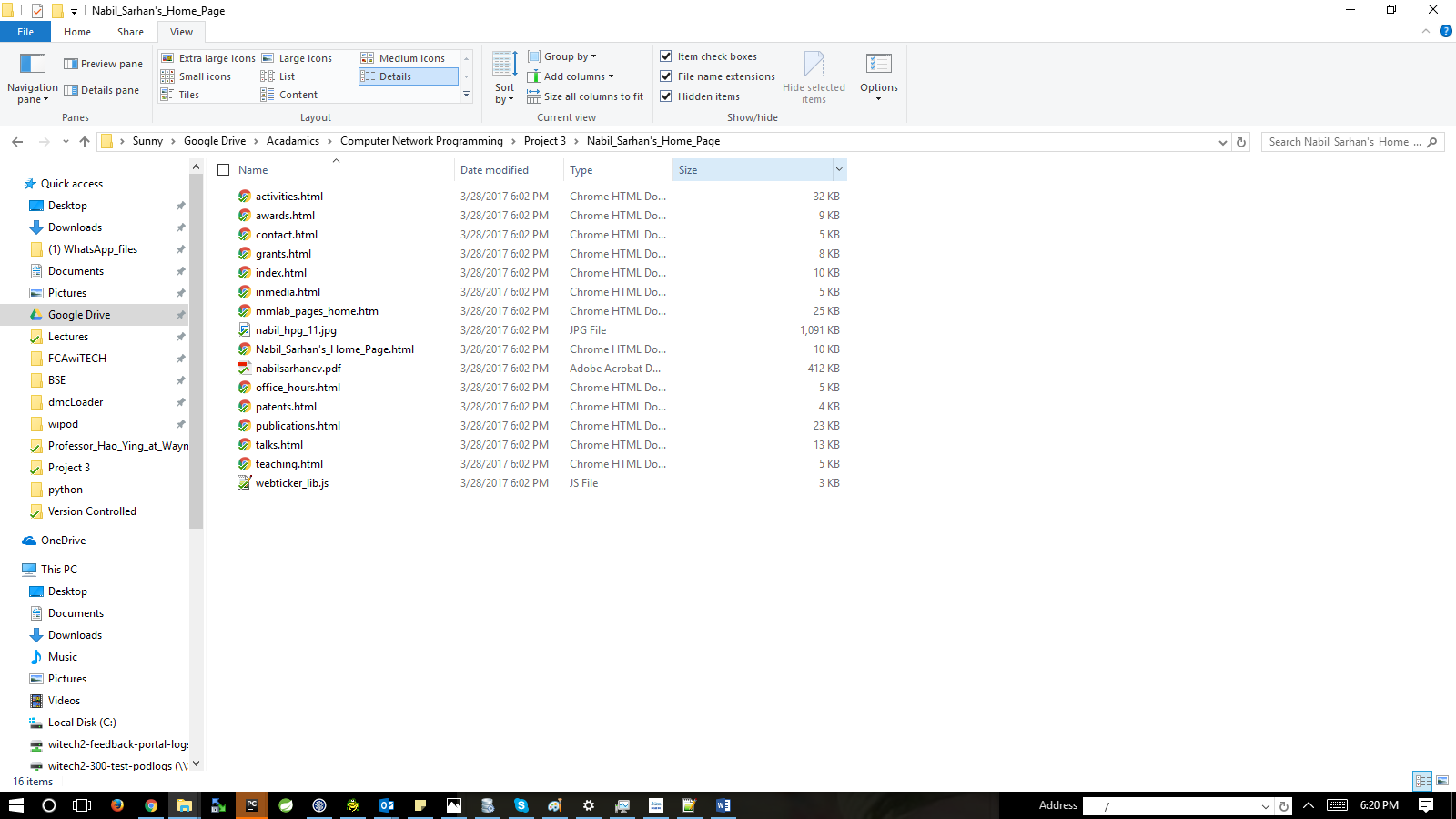
we are submitting project3.py and project2.py code but we are copying only project3.py code in this document to reduce length of the document

**URL #1:** [**http://www.ece.eng.wayne.edu/~nabil/**](http://www.ece.eng.wayne.edu/~nabil/)

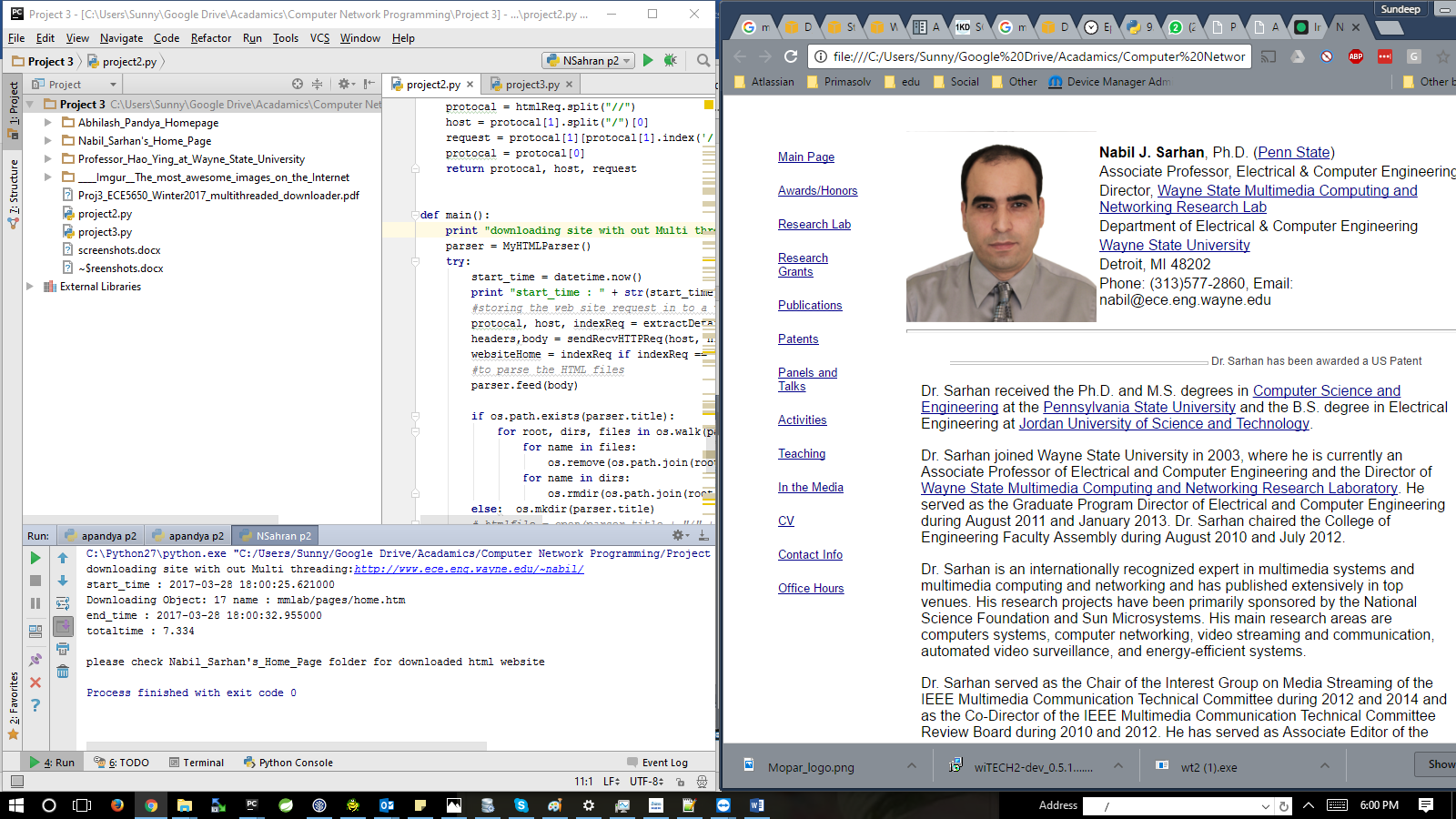
Max object size = 1091 KB

Min object size = 3 KB

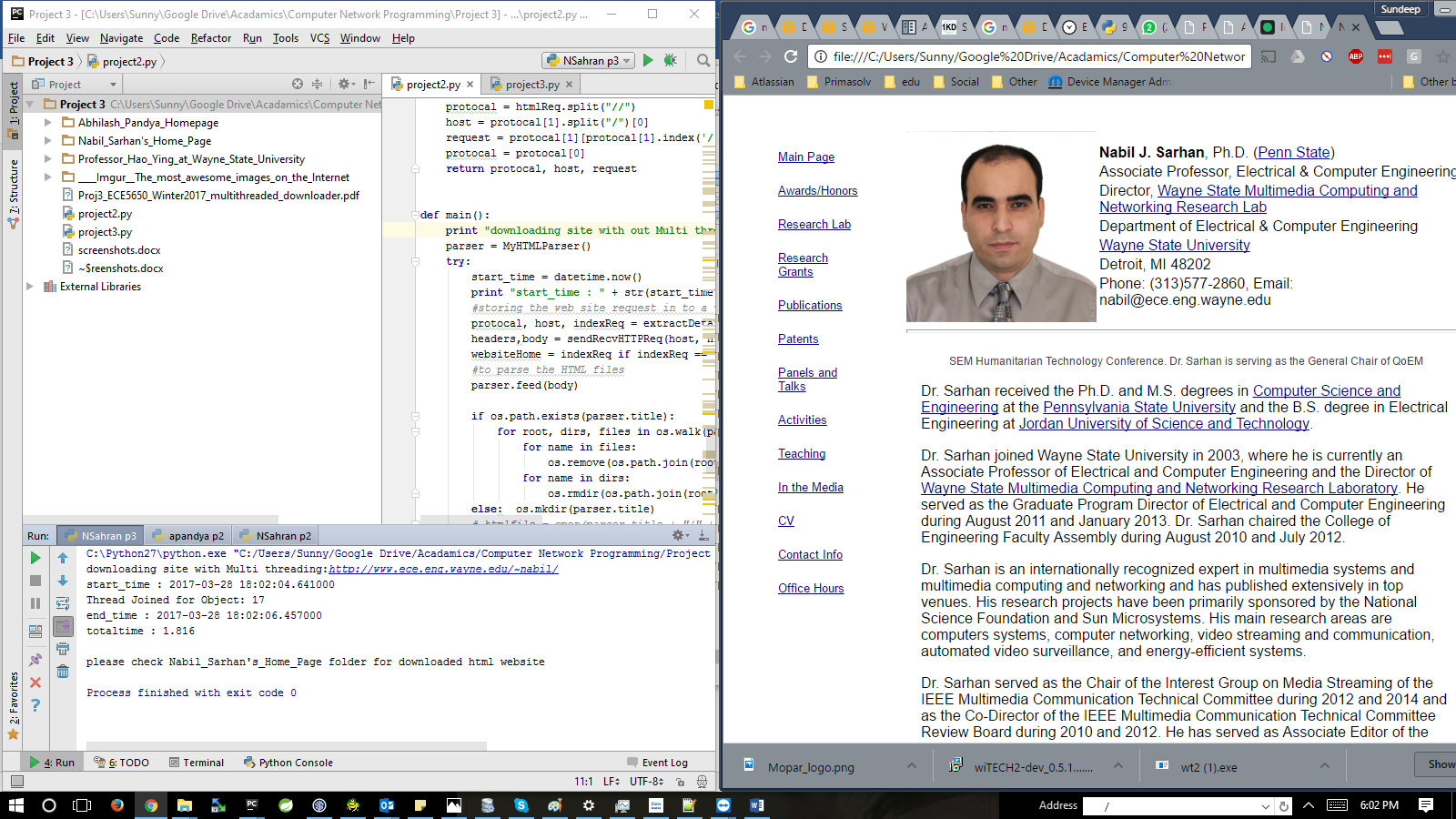
Items Downloaded:



Project 2 output



Project 3 Output

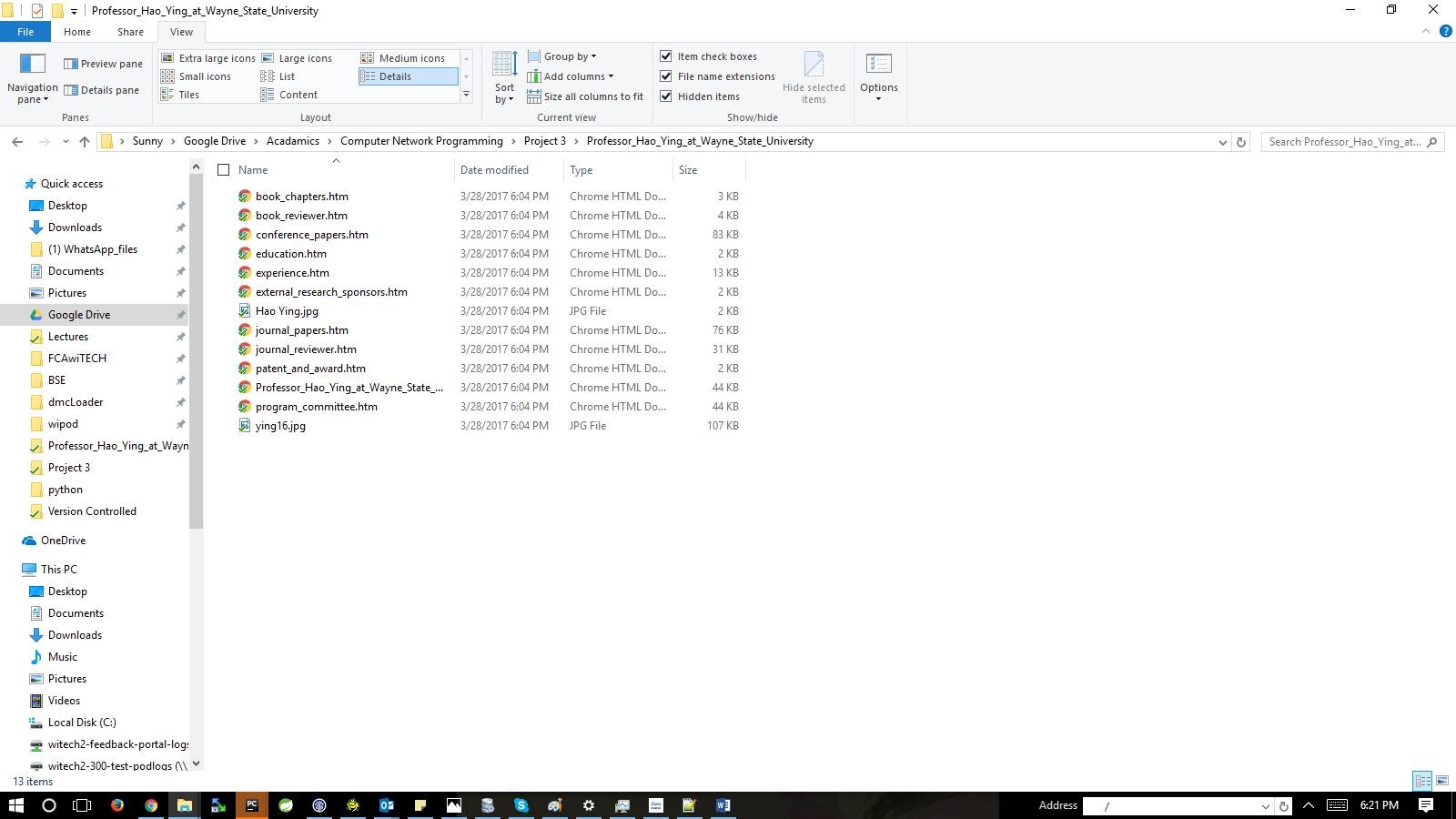


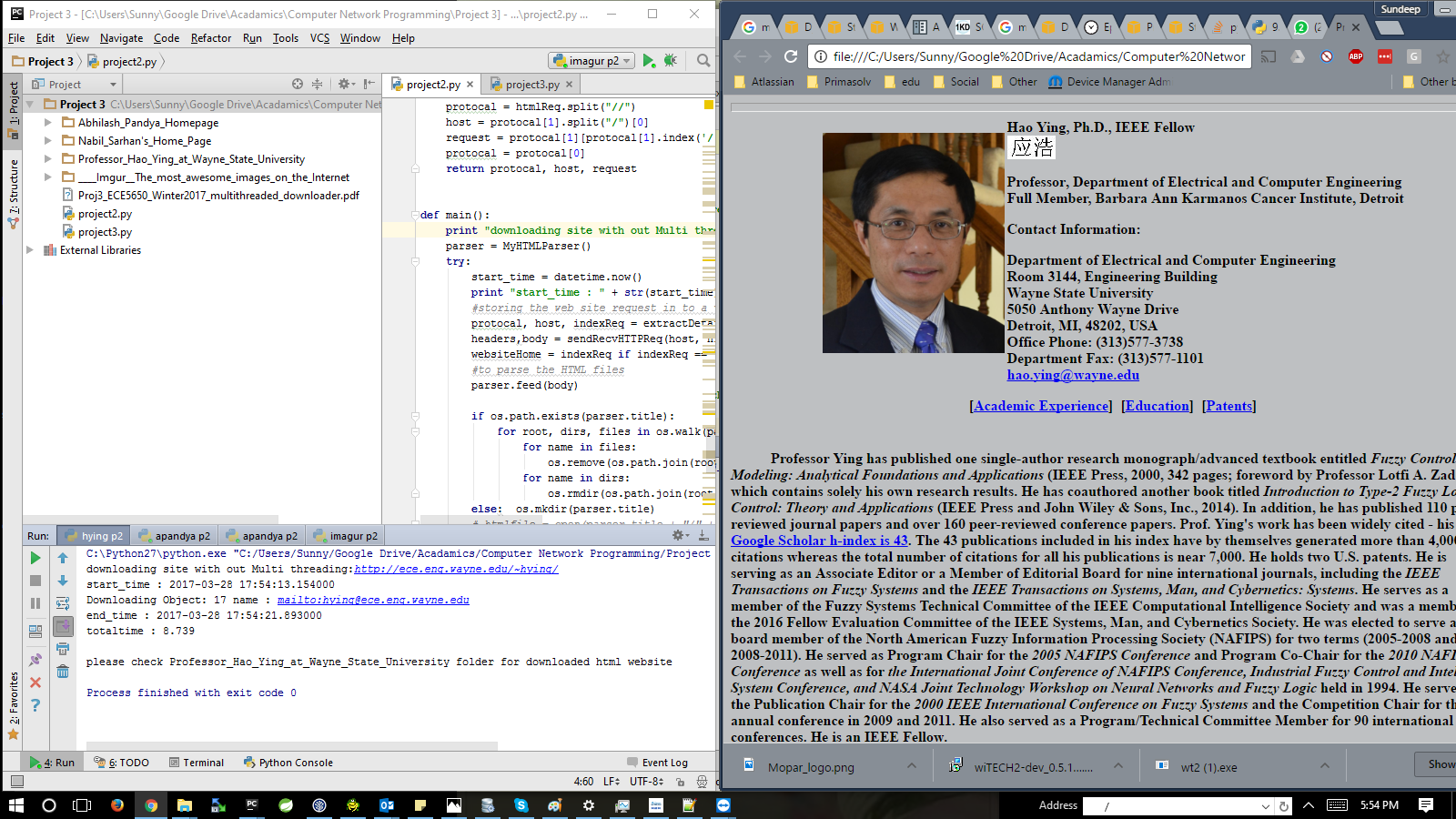
**URL #2:** [**http://ece.eng.wayne.edu/~hying/**](http://ece.eng.wayne.edu/~hying/)

Max object size: 107 KB

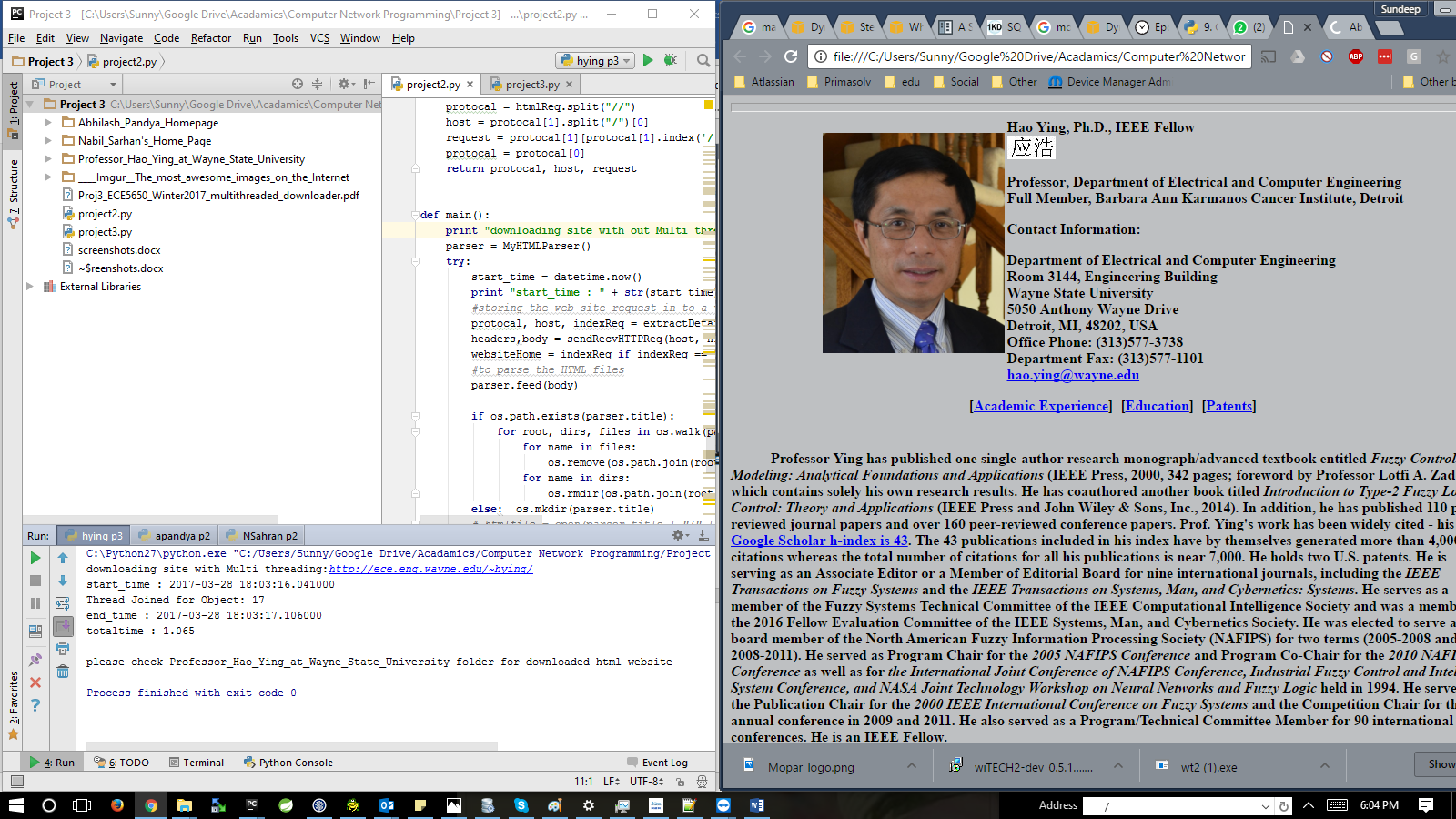
Min object size: 2 KB

Objects downloaded:



Project 2 output`

Project 3 output

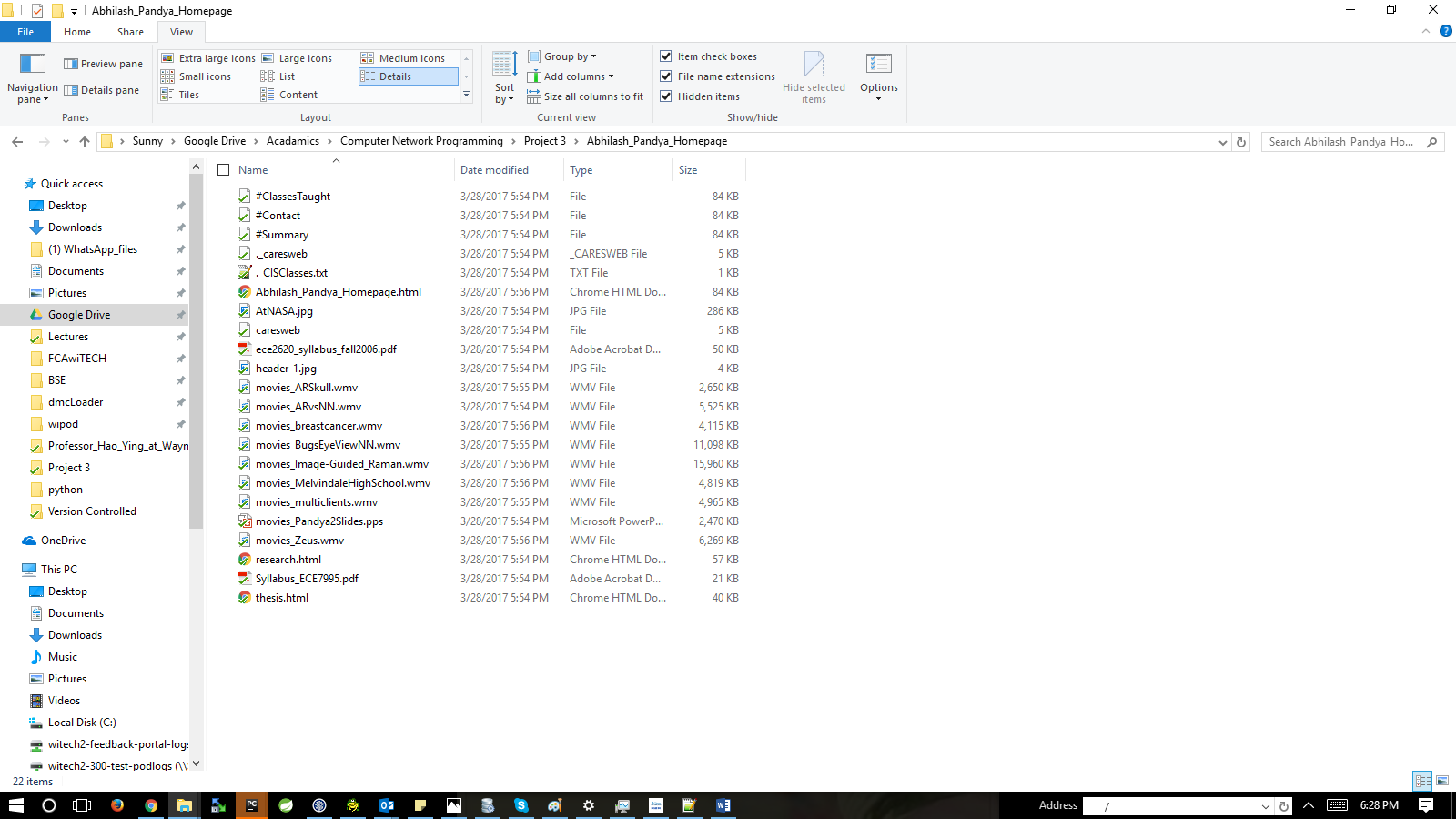


**URL #3:** [**http://ece.eng.wayne.edu/~apandya/**](http://ece.eng.wayne.edu/~apandya/)

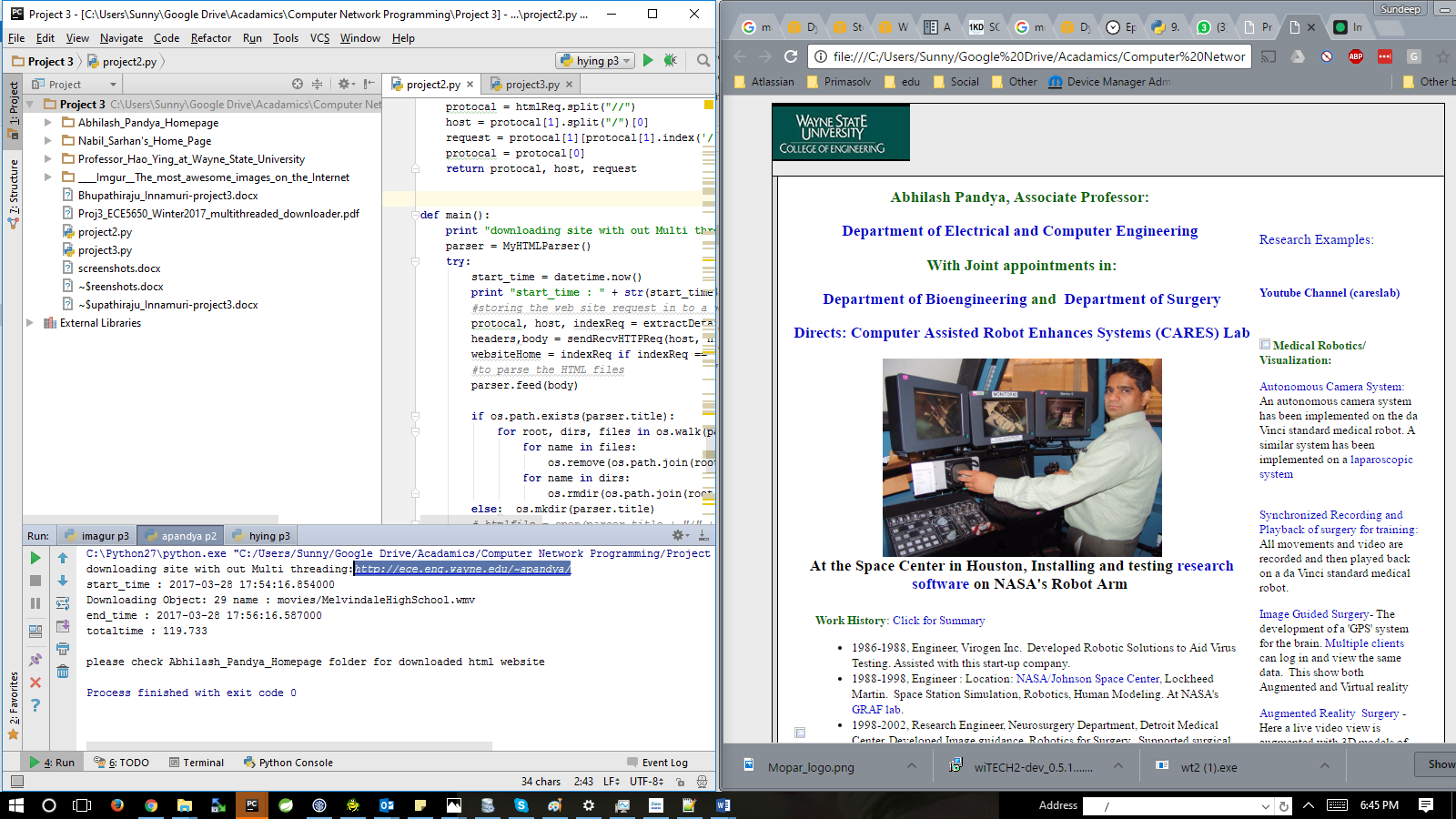
Max object size: 15960 KB

Min object size: 1KB

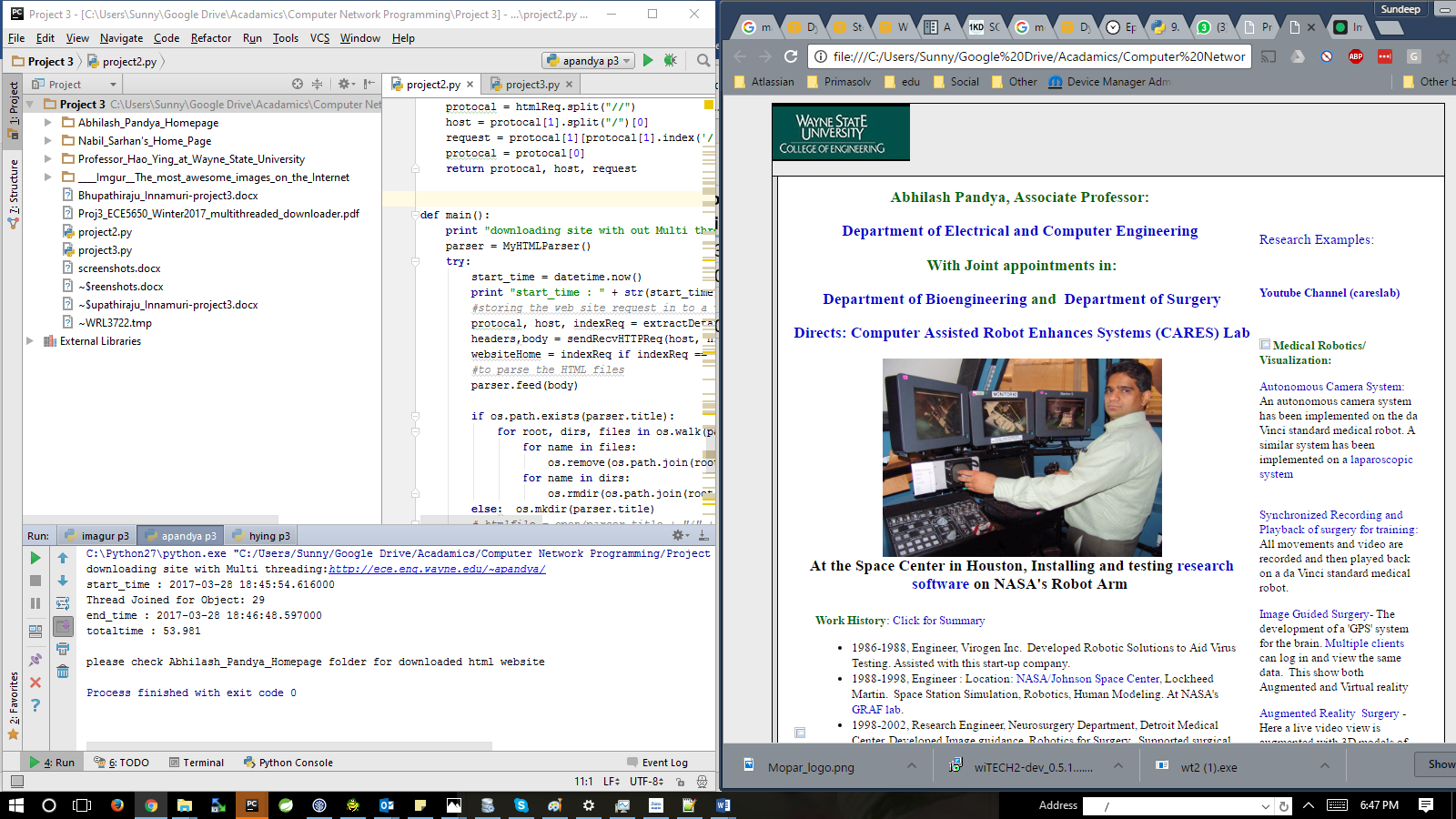
Items Downloaded:



Project 2 output



Project 3 output

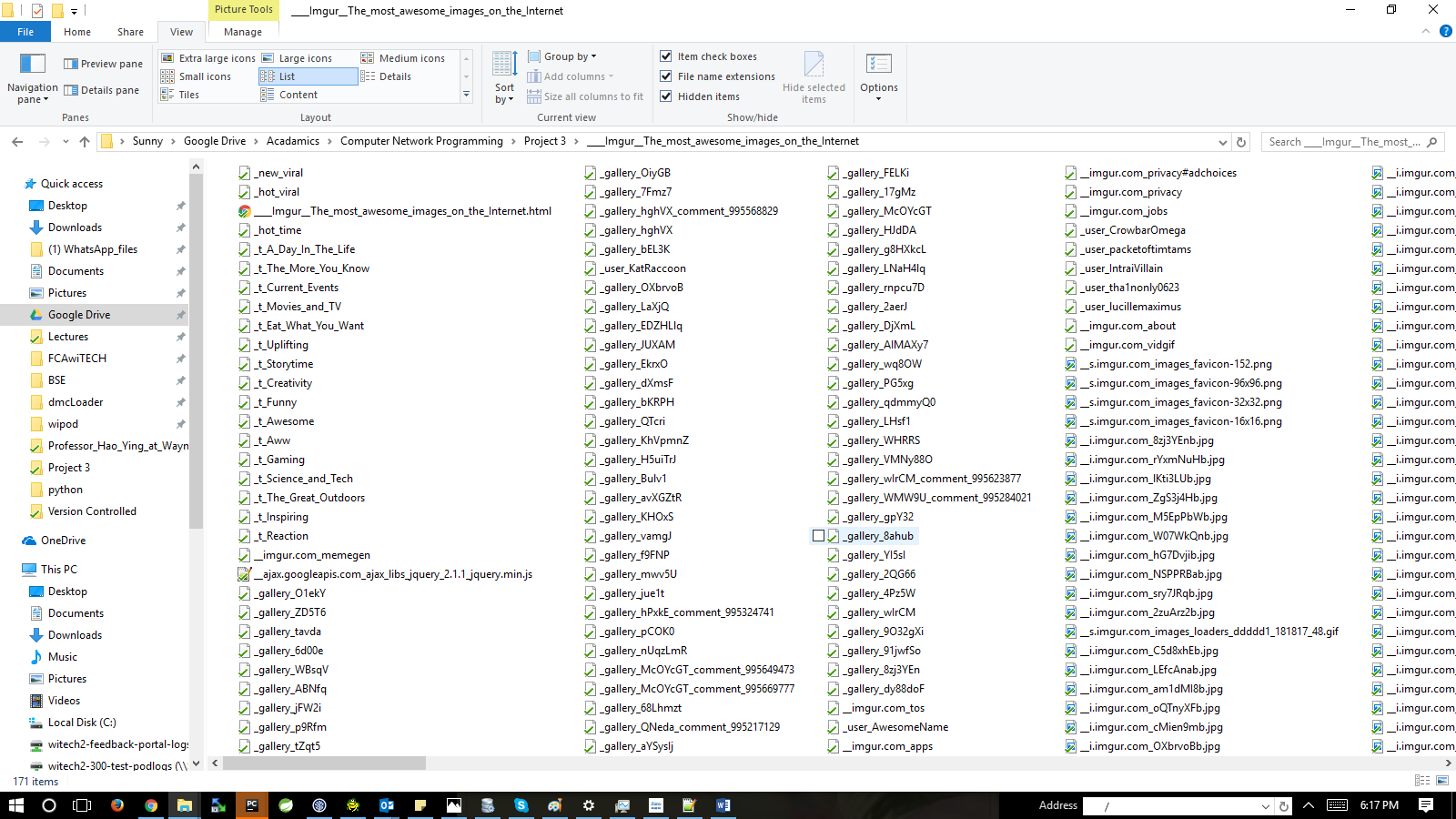


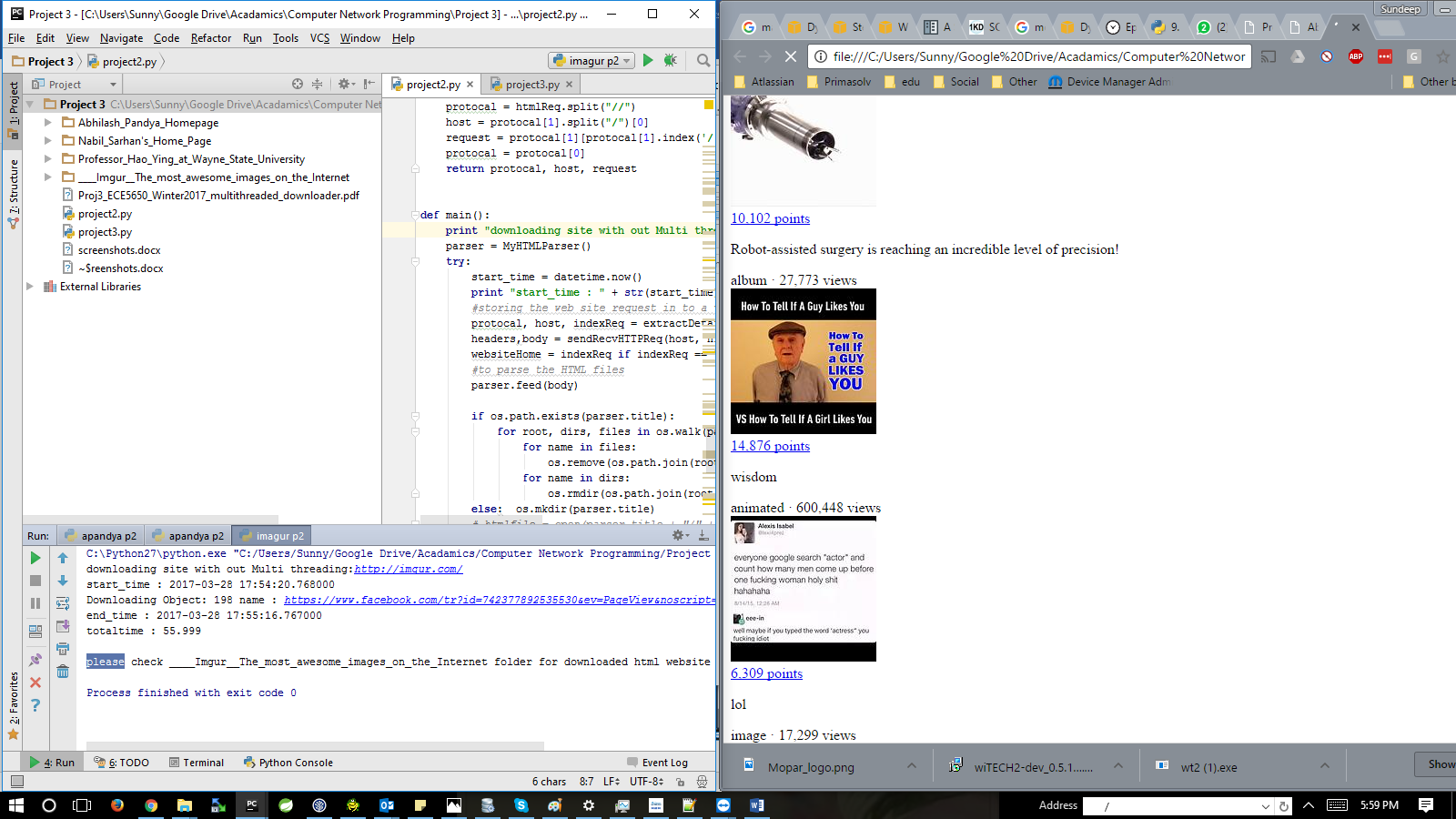
**URL:** [**http://imgur.com/**](http://imgur.com/)

Max object Size = 4 KB

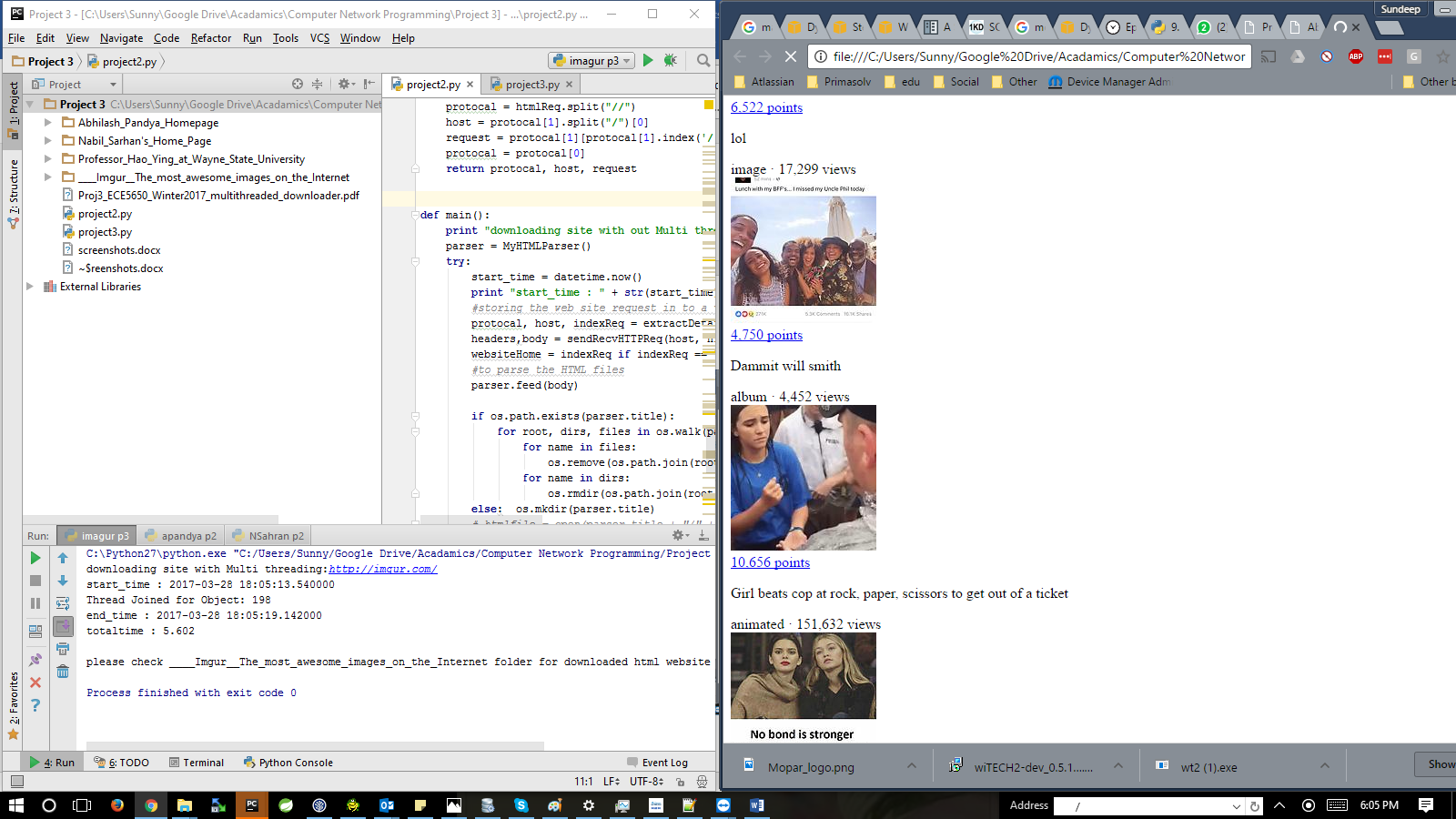
Min object size = 136 Kb

Items Downloaded:



Project 2 output

Project 3 output



**Analysis**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL # | No. of objects / No. of threads | downlaod time with out multi-threading(sec) | download time  with multi-threading(sec) | scaleup |
| 1 | 17 | 1.86 | 7.334 | 3.943 |
| 2 | 17 | 8.739 | 1.065 | 8.252 |
| 3 | 29 | 119.733 | 53.981 | 2.21 |
| 4 | 200 | 55.99 | 5.062 | 11.068 |

As, the webpages are different we get different speedups, we cannot derive a relationship between speedups, but we can try to understand why the speed ups differ.

We know that we can achieve best speedups for a parallel program when the task sizes are equal. Here we will try to compare the variance in task sizes with speedups i.e. the we will compare the variance in sizes objects downloaded with speedups

Variance will be given by

(max object size - min object size)/ min object size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| URL # | Max object size (KB) | Min object size(KB) | variance | Speedup obtained in above table |
| 1 | 1091 | 3 | 362.66 | 3.943 |
| 2 | 107 | 2 | 52.5 | 8.252 |
| 3 | 10960 | 1 | 10959 | 2.21 |
| 4 | 136 | 4 | 33 | 11.068 |

As we can see that the speed up decreases as the variance in object size increases.

Conclusion:

We have developed a multi-threading program to download web pages and calculated the performance, how and why different speedups are achieve due to the variance in object sizes